

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Линейные структуры данных: стек, очередь
Вариант 2

Студент гр. 8304

—

Птухов Д. А.

Преподаватель

—

Фиалковский М. С.

Санкт-Петербург

2019

Цель работы.

Получить опыт работы с очередью и стеком (реализация на основе массива).

Постановка задачи.

Содержимое заданного текстового файла F, разделенного на строки, переписать в текстовый файл G, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка как среди цифр, так и среди остальных литер строки). Реализация на основе очереди.

Описание алгоритма.

- 1) Считывание осуществляется при помощи конструкции:

```
while (std::getline(in, currentFileString))
```
- 2) После считывания очередной строки из файла F извлекаются все числа и заносятся в очередь.
- 3) После считывания всех чисел из очередной считанной строки осуществляется их запись в конец входной строки при помощи методов `pop` и `front` класса `Queue`.

Спецификация программы.

Программа предназначена для переноса всех чисел строк файла F в конец и записи полученных строк в файл G.

Программа написана на языке C++.

Описание функций и структур данных.

- 1) Для хранения значений в очереди был реализован шаблонный класс `Vector`.

Данный класс содержит 3 поля – `data_` - поле соответствующее данным, `size_` - поле соответствующее текущему значению массива,

capacity_ - поле соответствующее текущему размеру массива и методы, которые позволяют удалять элемент из массива по заданной позиции, добавлять элемент в массив, узнавать размер массива и получать первый элемент массива.

2) Для реализации очереди был создан шаблонный класс Queue.

Данный класс содержит поле data_ - поле для хранения данных и методы front, pop, push, empty, size которые реализуют интерфейс очереди.

Тестирование.

Таблица 1 – Результаты тестирования программы

Input	Output
wiognwiuti4gn38h76793h3g b3298h2	wiognwiutignhghgbh438767933 32982
f8327yhf92fh1hn98hg20g29 4h29	fyhffhnhhggh832792198202942 9
g287hg4b39h298392bi34343 u2bgier	ghgbhbiubgier287439298392343 432
19	19
1	1
a	a
((((((KJFIONO>>OPMOINI Ojf0j320>>>MMMM22<<<< <<<<<	((((((KJFIONO>>OPMOINIOj fj >>>MMMM<<<<<<<032022
f2nf29fn29gn92bg92g29hgb 291h rj109jr 1h	fnffngnbgghgbh rjjrh229299292292911091

Выводы.

В ходе работы был получен опыт работы с очередью на основе массива.

Приложение А. Исходный код программы.

Vecqu.cpp

```
#include "Vecqu.h"

using namespace vecqu;

std::string getStringNumberValue(std::string& currentFileString)
{
    auto numberBegin = std::find_if(currentFileString.begin(),
currentFileString.end(),
    [](char currentCheckElement) {return std::isdigit(currentCheckElement);
});
    if (numberBegin == currentFileString.end())
        return "";

    std::string stringNumberValue = "";
    auto numberEnd = numberBegin;

    while (numberEnd != currentFileString.end() && isdigit(*numberEnd))
    {
        stringNumberValue += *numberEnd;
        numberEnd++;
    }
    currentFileString.erase(numberBegin, numberEnd);

    return stringNumberValue;
}

int main(int argc, char** argv)
{
    Queue<long long> queue;

    if (argc > 2)
    {
        std::ifstream in(argv[1]);
        std::ofstream out(argv[2]);

        if (!in.is_open())
        {
            std::cout << "Input file is incorrect. Try to change data\n";
            return 0;
        }

        if (!out.is_open())
        {
            std::cout << "Output file is incorrect. Try to change data\n";
            return 0;
        }

        std::string currentFileString = "";

        while (std::getline(in, currentFileString))
        {
            if (*(currentFileString.end() - 1) == '\r')
                currentFileString.erase(currentFileString.end() - 1);
            std::cout << "Current check-string - " + currentFileString + "\n"
+ "Numbers in check-string in right order: ";

            while (1)
            {
                std::string stringNumberValue =
getStringNumberValue(currentFileString);
                if (stringNumberValue.empty())
                    break;

                long long numberValue = 0;
                try
                {
                    numberValue = stoll(stringNumberValue);
                }
            }
        }
    }
}
```

```

        catch (std::out_of_range)
        {
            out << "Entered number is out of range. Try to change
input data";
            break;
        }

        queue.push(numberValue);
        std::cout << numberValue << " ";
    }
    std::cout << std::endl << std::endl;
    while (!queue.empty())
    {
        currentFileString += std::to_string(queue.front());

        try
        {
            queue.pop();
        }
        catch (std::out_of_range& e)
        {
            std::cout << e.what() << std::endl;
        }
    }

    out << currentFileString << "\r\n";
}
std::cout << "Work complete let's eat!\n";
}

```

Vecqu.h

```
#pragma once
#include <iostream>
#include <exception>
#include <string>
#include <fstream>
#include <cctype>
#include <algorithm>

namespace vecqu
{
    template <typename T>
    T* resize(T* const& data, size_t old_size, size_t new_size)
    {
        T* new_data = new T[new_size];
        for (size_t i = 0; i < old_size; i++)
            new_data[i] = data[i];
        delete[] data;

        return new_data;
    }

    template <typename T>
    class Vector
    {
    public:
        Vector() = default;

        Vector(size_t size_) : size_(10), capacity_(20), data_(new
T[capacity_])
        { }

        ~Vector()
        {
            delete[] data_;
        }

        void erase(size_t eraseInd)
        {
            for (size_t i = eraseInd; i < size_; i++)
                data_[i] = data_[i + 1];
            size_--;
        }

        T front() const
        {
            return *data_;
        }

        size_t size() const
        {
            return size_;
        }

        void push_back(T pushArg)
        {
            size_++;
            if (size_ >= capacity_)
            {
                data_ = resize(data_, capacity_, capacity_ + 10);
                capacity_ += 10;
            }
        }
    };
}
```

```

        data_[size_ - 1] = pushArg;
    }

    bool empty() const
    {
        return size_ == 0;
    }

    T operator[](size_t ind) const
    {
        return data_[ind];
    }

    T& operator[](size_t ind)
    {
        return data_[ind];
    }

private:
    T* data_ = nullptr;
    size_t size_ = 0;
    size_t capacity_ = 0;
};

template <typename T>
class Queue
{
public:
    Queue() = default;

    void push(T pushArg)
    {
        queue_.push_back(pushArg);
    }

    void pop()
    {
        if (queue_.empty())
            throw std::out_of_range("Queue is empty!");

        queue_.erase(0);
    }

    T front() const
    {
        return queue_.front();
    }

    bool empty() const
    {
        return queue_.empty();
    }

    size_t size() const
    {
        return queue_.size();
    }

private:
    Vector<T> queue_;
};
}

```